Nicolas Omont
Cédric Boidin

9 Décembre 2002

Projet du cours « Reconnaissance de Formes »

Reconnaissance de mots manuscrits par détection de l'enveloppe convexe

Table des matières

1) Données

Un des problèmes de la reconnaissance de forme est d'extraire des informations pertinentes et discriminantes d'une grande quantité de données. Dans le cas que nous avons étudié, les données étaient des images résultant de la numérisation de mots écrits à la main. Ces images ont déjà été traitées. Elles sont donc peu bruitées, même si nous verrons que notre méthode est très sensible au bruit. En fait, ce sont des images à deux couleurs : noir ou blanc.

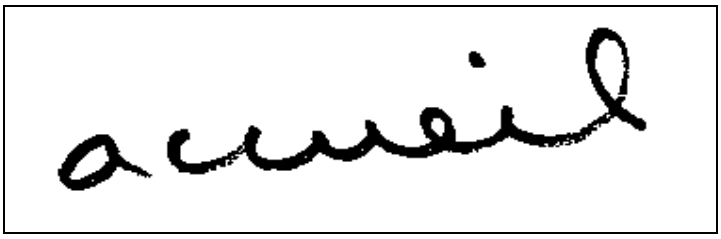


Figure 1 Exemple d'image traitée

Les mots contenus dans les images sont au nombre de 10:

accueil	auberge	bar	carte	chambre
douche	hotel	menu	relais	restaurant

2) Méthode

La méthode compte trois étapes : extraction des informations pertinentes, calcul de distances, et enfin classification.

Vu le faible nombre de mots à discriminer, on peut se permettre de n'extraire qu'une partie de l'information. On le fait en deux étapes :
-Comme ces images sont bicolores, on peut facilement les représenter par un ensemble de points, par exemple l'ensemble des points noirs. La seule information perdue est la taille de l'image, mais elle n'est pas pertinente.
-Ensuite, on détermine l'enveloppe convexe de l'ensemble des points, c'est là que la perte d'information est drastique. On ne conserve en fait que l'allure générale du mot, c'est à dire sa taille, et les lettres qui « dépassent » en dessous et au dessus de la ligne moyenne d'écriture.

Il faut ensuite comparer les chaînes. Pour pouvoir le faire, à la suite de normalisations éventuelles, on les code grâce au code de Freeman, ce qui permet ensuite de les aligner grâce un algorithme classique d'alignement de chaînes (programmation dynamique).

Enfin, un classifieur décide la classe d'appartenance de l'image étudiée. On utilise la méthode des k plus proches voisins.

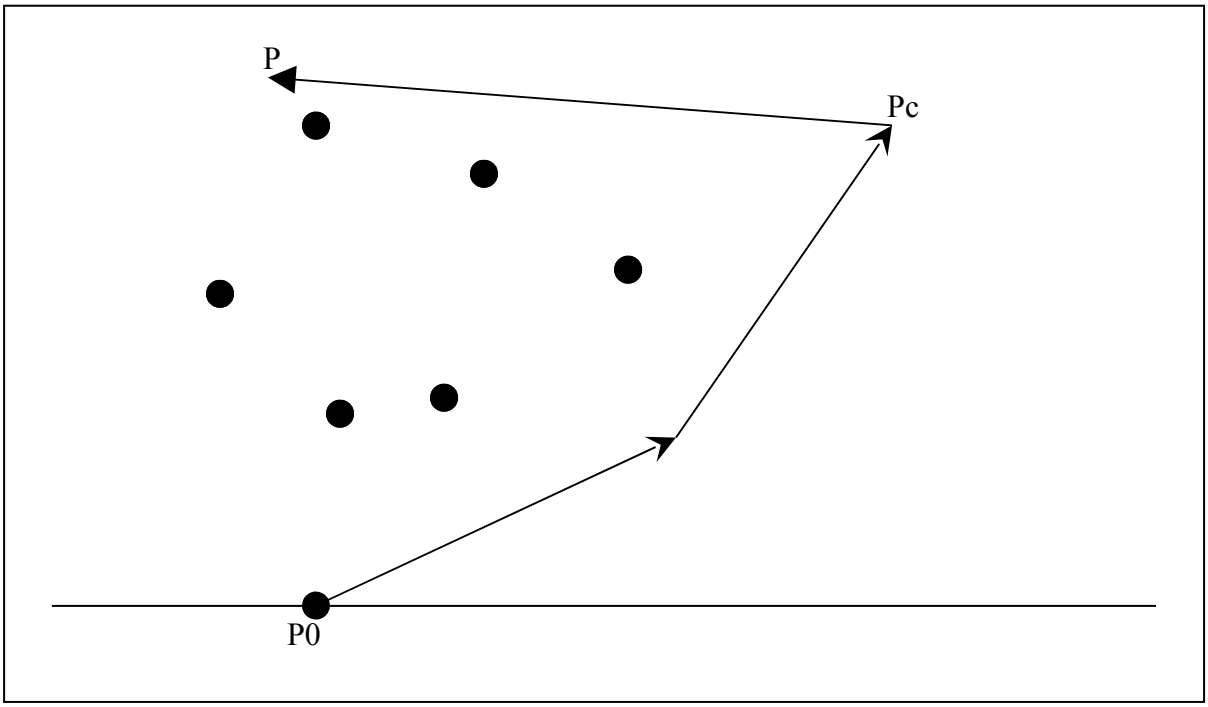
a) L'algorithme de détection de l'enveloppe convexe

Comme les images ne sont pas très volumineuses, on peut se permettre d'utiliser un algorithme simple de détection d'enveloppe convexe, bien qu'il soit en $O(n^2)$ alors que les meilleurs algorithmes sont linéaire par rapport au nombre de points dont on calcule l'enveloppe. Cet algorithme est détaillé dans « Algorithms » par R. Sedgewick[1]. Le principe en est le suivant :

- Trouver le point d'ordonnée minimale dans l'ensemble des points. Ce point P0 fait partie de l'enveloppe convexe.
- Faire
  - Affecter le point précédemment trouvé dans la boucle ou bien P0 au point courant Pc
  - Trouver le point P dans l'ensemble des points qui n'appartiennent pas à l'enveloppe convexe tel que l'angle du vecteur **PcP** avec l'horizontale soit minimum (L'angle est compris entre 0 et 360 degrés). Ce point appartient à l'enveloppe convexe

  Tant que l'angle minimum trouvé est inférieur à l'angle de **PcP0** avec l'horizontale.

La condition d'arrêt correspond au fait que le point suivant dans l'enveloppe convexe est le point d'ordonnée minimum P0.



Cet algorithme est très simple, il donne des résultats satisfaisants suffisamment rapidement sur les échantillons étudiés. Voici un exemple de ce que l'on obtient :

Figure 2 Exemple d'enveloppe convexe

[1] Robert **Sedgewick**: **Algorithms**. Addison-Wesley **1983**

On constate aussi que cet algorithme est très sensible au bruit. En effet le moindre point noir aberrant entraîne un résultat global complètement inutilisable :

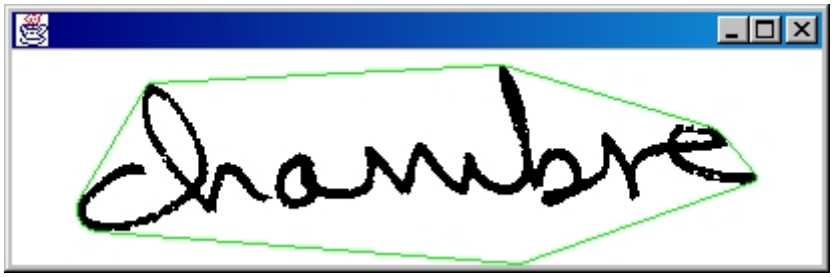


Figure 3 Enveloppe convexe aberrante

Ainsi, il est nécessaire de prévoir un filtrage très serré des images avant d'utiliser cet algorithme, quitte à perdre des points qui font vraiment partie de la trace écrite.

b) Codage de l'enveloppe

Pour pouvoir comparer les enveloppes entre elles, nous avons décidé de les coder en chaînes de symboles. Pour cela, nous avons utilisé le codage de Freeman. Le codage de Freeman consiste à découper chaque segment de l'enveloppe en plusieurs petits segments de longueurs et directions connues, symbolisés ensuite par des entiers.

Nous avons utilisé 8 symboles, les entiers de 0 à 7, comme le montre la figure suivante :

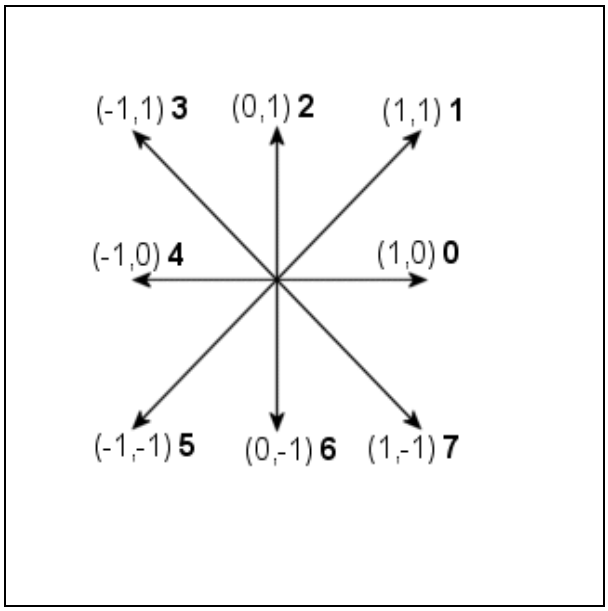


Figure 4 Codage de Freeman

Le codage d'un segment s'effectue par récurrence. On calcule d'abord l'angle que fait le segment avec l'horizontale. Le premier symbole sera celui dont l'angle se rapproche le plus de l'angle du segment.
On déplace ensuite l'extrémité du segment dans la direction du symbole (on utilise les coordonnées inscrites entre parenthèses sur le schéma), puis on recommence, jusqu'à coder tout le segment.
On réitère le processus pour tous les segments. La suite des symboles de chaque segment donne la chaîne de l'enveloppe convexe.

La figure suivante montre un exemple de codage. Les dimensions ont été amplifiées pour que les deux lignes soient distinctes.

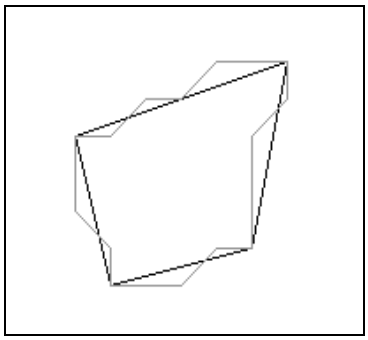


Figure 5 Exemple de codage de Freeman

En noir, l'enveloppe réelle ; en gris, l'enveloppe codée.
En partant du coin en haut à gauche, la chaîne codée est : 6676001022212445454

c) Comparaison de deux enveloppes

La comparaison de deux chaînes s'effectue par programmation dynamique. Pour cela, il faut définir les coûts de suppression, d'insertion et de substitution.

Le coût de substitution a été défini comme la différence entre 2 symboles :

```
int coutdiag=Math.min(   Math.abs(seq1[i]-seq2[j]),
                         8-Math.abs(seq1[i]-seq2[j]));
```

Cette formule permet de prendre en compte tous les cas possibles. La valeur de ce coût vaut un entier compris entre 0 (même symbole : pas de substitution) et 4 (symboles opposés).
Les coûts de suppression et d'insertion sont égaux. Leur valeur est un paramètre à choisir pour avoir une bonne segmentation.

Le tableau de la page suivante fournit un exemple de programmation dynamique entre les deux chaînes en gras (avec un coût d'insertion-suppression de 2). Le résultat de la comparaison est donné par la valeur du dernier point du tableau. Cette valeur est une sorte de distance entre les deux chaînes : si sa valeur est faible, les deux chaînes se ressemblent, si la valeur est élevée, les deux chaînes sont éloignées.

	6	***6***	***7***	***6***	***0***	***0***	***1***	***0***	***2***	***2***	***2***	***1***	***2***	***4***	***4***	***5***	***4***	***5***	***4***
7	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36
7	2	1	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34
0	4	3	2	4	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
7	6	5	3	3	5	5	7	9	11	13	15	17	19	21	23	25	27	29	31
0	8	7	5	5	3	5	6	7	9	11	13	15	17	19	21	23	25	27	29
2	10	9	7	7	5	5	6	8	7	9	11	13	15	17	19	21	23	25	27
2	12	11	9	9	7	7	6	8	8	7	9	11	13	15	17	19	21	23	25
2	14	13	11	11	9	9	8	8	8	8	7	9	11	13	15	17	19	21	23
1	16	15	13	13	11	10	9	9	9	9	9	7	9	11	13	15	17	19	21
2	18	17	15	15	13	12	11	11	9	9	9	9	7	9	11	13	15	17	19
3	20	19	17	17	15	14	13	13	11	10	10	11	9	8	10	12	14	16	18
3	22	21	19	19	17	16	15	15	13	12	11	12	11	10	9	11	13	15	17
4	24	23	21	21	19	18	17	17	15	14	13	14	13	11	10	10	11	13	15
3	26	25	23	23	21	20	19	19	17	16	15	15	15	13	12	12	11	13	14
4	28	27	25	25	23	22	21	21	19	18	17	17	17	15	13	13	12	12	13
6	30	28	27	25	25	24	23	23	21	20	19	19	19	17	15	14	14	13	14
6	32	30	29	27	27	26	25	25	23	22	21	21	21	19	17	16	16	15	15
6	34	32	31	29	29	28	27	27	25	24	23	23	23	21	19	18	18	17	17
5	36	34	33	31	31	30	29	29	27	26	25	25	25	23	21	19	19	18	18
6	38	36	35	33	33	32	31	31	29	28	27	27	27	25	23	21	21	20	20

Tableau de programmation dynamique.
Le chemin en gris est le chemin optimal.
La distance entre les 2 chaînes est 20.

3) Résultats

Le programme tel que nous l'avons décrit fournit de très bons résultats pour la base de mots qui nous a été fournie (après suppression des quelques points aberrants de quelques images). Nous avons utilisé 10 classes de mots, chacune possédant 6 échantillons.

Nous avons donc effectué l'extraction des enveloppes convexes puis le codage de toutes les enveloppes. Ensuite, nous avons sélectionné une enveloppe de test que nous avons comparée aux 59 autres. Les distances obtenues ont été triées. Théoriquement, les enveloppes de la même classe que celle testée devraient fournir des distances plus faibles que les autres enveloppes.

Par exemple, les résultats obtenus pour l'échantillon test « accueil1.gif » sont :

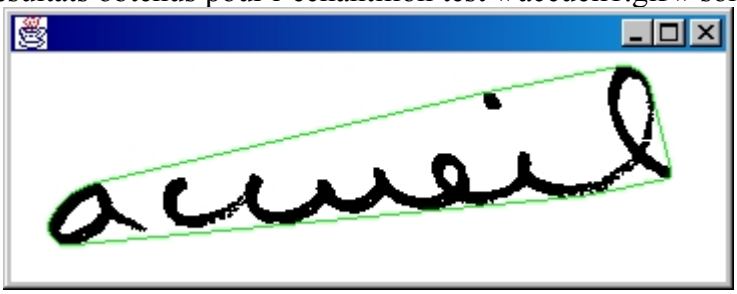


Figure 6 Enveloppe convexe de accueil1.gif

Enveloppe en points :

(23,76) (29,70) (32,68) (38,65) (237,19) (304,6) (309,6) (311,7) (313,8) (314,9) (315,10) (316,11) (318,14) (319,17) (329,58) (329,59) (329,60) (329,61) (328,62) (289,70) (283,71) (31,95) (25,95) (23,94) (19,90) (18,87) (18,84)

Enveloppe codée :

777676767777777707070700000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000700707007070070707007070070707007070070707007070070
707007070070707007070700707070070700707070070707007070070707007070070707007070000
000000000000000000000000000000000000700707070070707007070707007070070707000000101011
111221222222222222222222221221212212121221212212122223444444444444444444444344
3434344343443434444434444444444444444444444444444444444444444444444444444444
444444444444444444444444444444444444444444444444444444444444444444444444444444
444444444444444444444444444444444444444444444444444444444444444444434343443
4344343434434344343443434344343443434344343443434444444454555656666

Classement des 10 chaînes les plus ressemblantes, avec leurs distances respectives.

1 (accueil5, 111)
2 (accueil6, 111)
3 (accueil3, 115)
4 (accueil4, 138)
5 (chambre4, 148)
6 (accueil2, 149)
7 (douche6, 164)
8 (menu2, 165)
9 (douche2, 178)
10 (chambre6, 185)

Les autres échantillons de la classe « accueil » arrivent très bien classés, en positions 1, 2, 3, 4, 6. Par conséquent, si on classe l'échantillon de test dans la classe de l'échantillon le plus proche, cet exemple est un succès.
Ainsi, si l'on effectue ce même processus pour tous les échantillons, en les comparant aux 59 autres échantillons, on n'enregistre que 2 erreurs, soit un taux de réussite de 96.7%.

Une exemple de confusion : pour l'échantillon test hotel6, l'échantillon le plus proche est menu4 (à une distance de 111), le premier échantillon de la classe hotel est l'échantillon hotel4 (à une distance de 212).

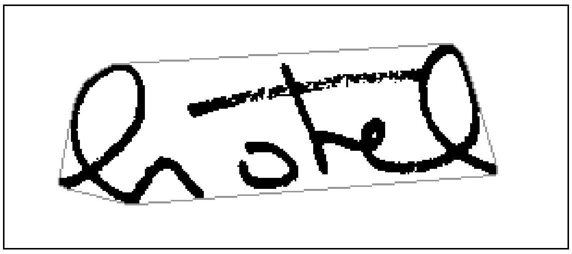


Figure 7 échantillon test : hotel6

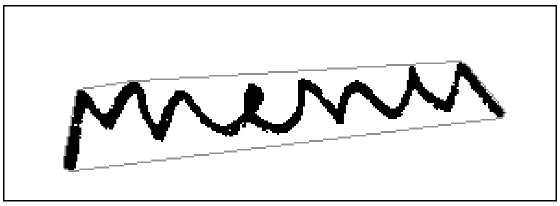


Figure 8 plus proche voisin : menu4

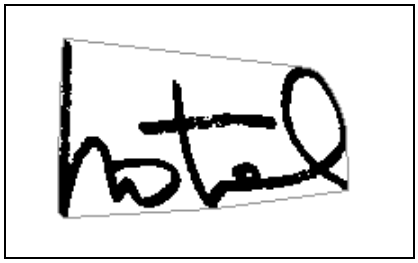


Figure 9 « hotel » le plus proche : hotel4

4) Tentatives d'amélioration

Nous avons effectué trois tentatives d'amélioration :
-une opération de rotation,
-une normalisation de la taille des mots,
-l'utilisation d'un classifieur de type k-plus proches voisins.

a) Rotation

L'opération de rotation a pour but de redresser les mots, pour qu'ils aient tous la même orientation et qu'ils deviennent ainsi comparables.
Notre idée a été de rendre horizontale la plus grande distance entre 2 points de l'enveloppe. Comme le montrent les exemples ci-dessous, cette méthode donne de très mauvais résultats pour les mots courts, et elle peut, dans certains cas, éloigner deux échantillons d'une même classe. Cette fonction ne sera plus utilisée par la suite.

Figure 10 Exemples d'enveloppes redressées

Figure 11 Exemple d'enveloppe mal redressée
la longueur maximale a été mesurée entre le haut du « h » et le bas du « l ».

b) Normalisation de la taille des mots

Pour pouvoir comparer les échantillons, nous avons décidé de normaliser la taille des mots. Nous avons ramené toutes les hauteurs à un indice 100, et dilaté les largeurs proportionnellement (pour garder le rapport hauteur/largeur).

c) Utilisation d'un classifieur de type k-plus proches voisins

Une autre idée d'amélioration a été l'utilisation d'un classifieur de type k-plus proches voisins. Pour trouver la classe de l'échantillon, nous avons pris la classe majoritaire parmi les k (k variant de 1 à 10) plus proches voisins.

Pour visualiser les résultats en fonctions des différents paramètres de l'algorithme, nous les avons rassemblés sur deux graphiques.

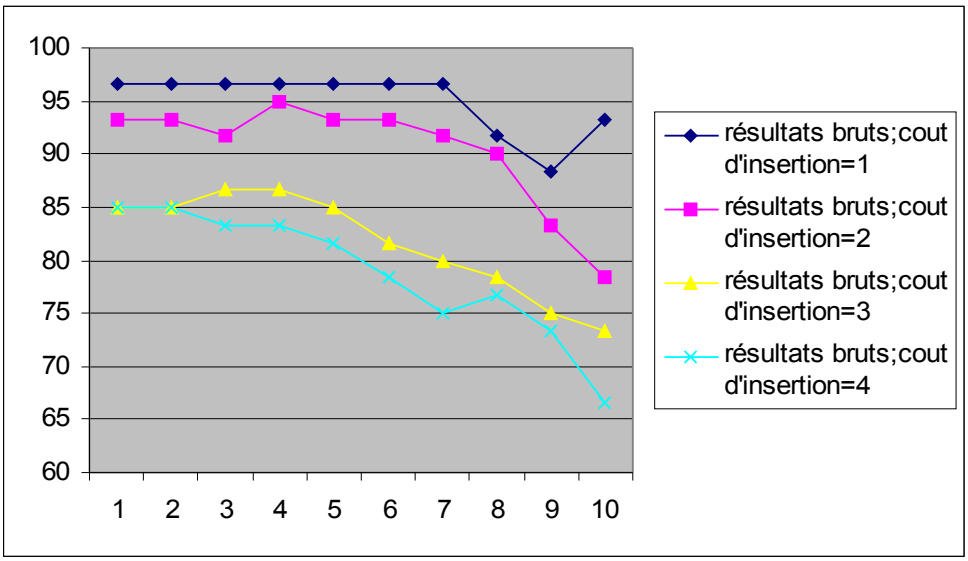


Figure 12 Taux de réussite en fonction du nombre k du classifieur, pour les enveloppes brutes

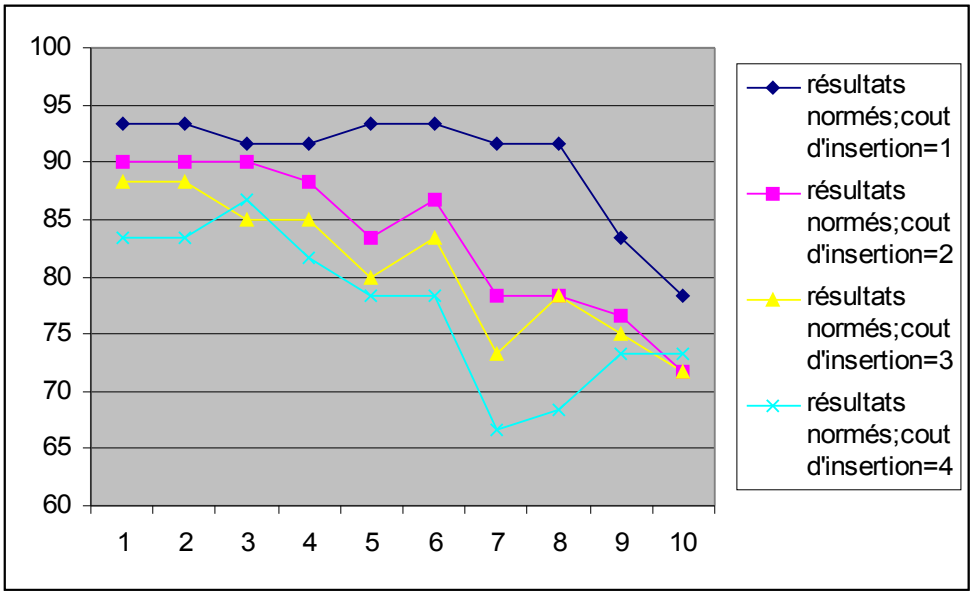


Figure 13 Taux de réussite en fonction du nombre k du classifieur, pour les enveloppes normées

Ces courbes ne sont pas concluantes quant aux résultats de nos tentatives améliorations. Les meilleurs résultats sont ceux obtenus sans normalisation, en considérant pour le classifieur la classe du (1-)voisin le plus proche.

5) Conclusion

La méthode de détection et de codage de l'enveloppe convexe semble permettre de classifier convenablement des mots. Les résultats sur nos échantillons sont très bons. La qualité de ces résultats s'explique notamment par la qualité des échantillons. Ces échantillons étant de bonne qualité, la normalisation que nous avons testée n'améliore pas les résultats. Nos autres tentatives d'amélioration (rotation, k-plus proches voisins), se sont révélées elles aussi infructueuses. Nous pensons que le moyen le plus probable d'amélioration et d'extension de l'algorithme est l'augmentation du nombre d'échantillons de la base d'apprentissage.

Annexe 1 : Structure du programme

L'algorithme a été programmé en java. Il comporte 3 fichiers : `tst.java`, `comparaison.java`, `rdf.java`.
Les classes utilisées sont les classes `chaine`, `tst`, `point`, `codage`, `comparaison`.

`tst.java :`

La classe `chaine` est un objet, qui contient le nom du fichier, ainsi que la chaîne de symboles correspondant à l'enveloppe du mot.

La classe `tst` comporte les fonctions de plus haut niveau : lancement du programme (`main`), codage des fichiers (`create_base`), calcul et tri des distances d'un échantillon à tous les autres échantillons de la base (`scores`), choix du classifieur (`classifieur`).

`comparaison.java :`

La classe `point` est un objet, un point ayant 2 coordonnées x et y.

La classe `codage` comporte les fonctions de codage de l'enveloppe en chaîne de symboles : codage de l'enveloppe (`codage`), codage d'un segment (`segment_codage`), calcul du symbole de freeman (`freeman`), normalisation de la taille de l'enveloppe (`norme`), rotation de l'enveloppe (`redresse`).

La classe `comparaison` comporte la fonction de comparaison entre deux enveloppes par programmation dynamique (`comparaison`).

`Rdf.java :`

La classe `rdf` comporte les fonctions de calcul et d'affichage de l'enveloppe convexe. La fonction `algorithm` extrait l'enveloppe convexe d'une image à deux couleurs. Elle fait appel à la fonction `convexHullAlg` qui extrait l'enveloppe convexe d'une liste de points. Le fichier contient une fonction `main`, ce qui permet d'afficher une enveloppe convexe à l'aide de la fonction `paint`. Ces fonctions de dessin sont inspirées de l'ouvrage de Whelan et Molloy[2].

[2] P.F. Whelan, D. Molloy, Machine Vision Algorithm in Java, Springer, 2001

Annexe 2 : Code source

tst.java

```
import java.io.*;
import java.awt.*;
import java.awt.image.*;
import javax.swing.*;

class chaine{
      String name;
      int[] seq;
      int lseq;
      int score;   //pour les comparaisons

      //création de la chaine à partir du nom de fichier
      chaine(String n) throws IOException {
            System.out.println(n);

            name=n;
            Rdf eg = new Rdf();
            BufferedImage input= eg.loadImage("./data/"+n+".gif");

            point[] hull=eg.algorithm(input);

            //possibilité d'afficher l'image et son enveloppe
            /*input = eg.paint(input, hull);
            ConvexHullCanvas chc=new ConvexHullCanvas();
            eg.getContentPane().add(new JLabel(new ImageIcon(input)));
            eg.pack();
            eg.setVisible(true);*/

            seq= new int[10000];  //Attention à la taille max du tableau
            lseq=codage.codage(hull,seq,0);

            //possiblité d'éditer les points et les codes
            /*BufferedWriter ecrire =new BufferedWriter(new
FileWriter("./txt/"+n+".txt"));//attention au répertoire
            for (int i=0;i< hull.length;i++) ecrire.write ("(" + hull[i].x
+","+ hull[i].y +") ");
            ecrire.write("\r" + hull.length +";" +lseq + "\r");
            ecrire.flush();
            for (int i=0;i< lseq;i++) ecrire.write (seq[i]+"");
            ecrire.flush();*/
      }

      chaine(){}

      chaine copy(){
            chaine c= new chaine();
            c.name=name;
            c.seq=seq;
            c.lseq=lseq;
            c.score=score;
            return c;
      }

}

/*******************************************************/

class tst{
```

```
	public static void main(String args[]) throws IOException{
		System.out.println("Execution");

		chaine[] base =create_base();
	//for (int coutdroit=4;coutdroit<5;coutdroit++){
	//BufferedWriter resultats =new BufferedWriter(new FileWriter("./txt/
resultatsbrut"+rn+".txt"));

	int nbclass=1;
	int coutdroit=1;
	//for (nbclass=1; nbclass<11; nbclass++){

		int cpt=0;
		for (int j=0;j<base.length;j++){
			chaine[] scores=scores(base,j,coutdroit);

			//possibilité d'écrire les résultats dans un fichier
			/*BufferedWriter ecrire =new BufferedWriter(new
FileWriter("./txt/resultats"+base[j].name+".txt"));
			for (int i=0;i< scores.length;i++) ecrire.write (i+"(" +
scores[i].name +","+ scores[i].score +")\r");
			ecrire.flush();*/

			System.out.println( base[j].name + "\t"
+classifieur(scores,nbclass));
			if
(classifieur(scores,nbclass).equals(base[j].name.substring(0,base[j].name.l
ength()-1))) cpt++;

		}
		double stat=(double)cpt/(double)base.length*100;
		System.out.println(stat);
		//resultats.write(nbclass + "\t" + stat + "\r");
		//resultats.flush();

	//}
	//}

		System.out.println("End");
	}

	/*******************************************************/

	//cree le tableau de chaines
	static chaine[] create_base()throws IOException{
		int n=6;
		chaine[] base=new chaine[10*n];
		for (int i=0;i<n;i++){
			base[i]=new chaine("accueil"+(i+1));
			base[i+n]=new chaine("auberge"+(i+1));
			base[i+2*n]=new chaine("bar"+(i+1));
			base[i+3*n]=new chaine("carte"+(i+1));
			base[i+4*n]=new chaine("chambre"+(i+1));
			base[i+5*n]=new chaine("douche"+(i+1));
			base[i+6*n]=new chaine("hotel"+(i+1));
			base[i+7*n]=new chaine("menu"+(i+1));
			base[i+8*n]=new chaine("relais"+(i+1));
			base[i+9*n]=new chaine("restau"+(i+1));
		}
		return base;
	}
```

```
      /*********************************************************/

      //base triée selon les scores de comparaison avec base[test]
      static chaine[] scores(chaine[] base, int test, int coutdroit) throws
IOException{
            chaine[] scores=new chaine[base.length-1];
            int j=0;

            //recopie du tableau dans le tableau scores
            for (int i=0;i<base.length;i++){
                  if (i!=test) {
                        scores[j]=base[i].copy();
                        j++;
                  }
            }

            //calcul des scores de comparaison
            for (int i=0;i<scores.length;i++){


scores[i].score=comparaison.comparaison(base[test].seq,base[test].lseq,scor
es[i].seq,scores[i].lseq, coutdroit);
            }

            tri(scores);
            return scores;
      }

      /*********************************************************/

      //tri à bulles
      public static void tri(chaine[] base) {
            int   i, j, end;

            for (i = 0, end = base.length; i < base.length - 1; ++i) {
            for (j = 1; j < end; ++j) {
                  if (base[j-1].score > base[j].score){
                  swap(base,j-1,j);
                        }
            }
            }
    }

      /*********************************************************/

      public static void swap(chaine[] base,int i,int j){
            chaine k=base[i].copy();
            base[i]=base[j];
            base[j]=k;
      }

      /*********************************************************/

      //classifieur de type k-ppv, k=n
      public static String classifieur(chaine[] scores, int n){
            int[] nbr= new int[n];
            String[] str= new String[n];
            int m=0;

            //calcule le nombre d'occurrences pour chaque classe
            for (int i=0; i<n;i++){
                  String
s=scores[i].name.substring(0,scores[i].name.length()-1);
                  int j=0;
```

```
                  while(j<m){
                        if (s.equals(str[j])) {
                              nbr[j] ++;
                              break;
                        }
                        j++;
                  }
                  if (j==m){
                        str[m]=s;
                        nbr[m]=1;
                        m++;
                  }
            }

            //rend la classe la plus représentée
            int max=0;
            String s="";
            for (int i=0;i<m;i++){
                  if(nbr[i]>max){
                        s=str[i];
                        max=nbr[i];
                  }
            }
            return s;
      }
}

/*********************************************************/
```

comparaison.java

```
import java.io.*;

class point{
      int x,y;

      point(int i1, int i2){
            x=i1;
            y=i2;
      }

      boolean equal(point p2){
            if (x==p2.x && y==p2.y) return true;
            return false;
      }

      void translate(int dx, int dy){
            x=x+dx;
            y=y+dy;
      }

      point copy(){
            return new point(x,y);
      }
}

/*********************************************************/

class codage
{
      //codage de l'enveloppe
      public static int codage(point[] p, int[] t, int n){  //modifie le
tableau de points
```

```
            //possibilité de normer l'enveloppe
            //norme(p);

            point pdebut=p[0].copy();
            for (int i=0;i<p.length-1;i++){
                  n=segment_codage(p[i],p[i+1],t,n);
            }
            n=segment_codage(p[p.length-1],pdebut,t,n);
            return n;
      }

      /*********************************************************/

      //codage d'un segment
      public static int segment_codage(point p1, point p2, int[] t, int n){
            if (p1.equal(p2)){
                  return n;
            }
            else{
                  int[] f=freeman(p1,p2);
                  t[n]=f[0];
                  n++;              //Attention à la taille du tableau
                  p1.translate(f[1],f[2]);
                  int m=segment_codage(p1,p2,t,n);
                  return m;
            }
      }

      /*********************************************************/

      //détermination du symbole de freeman
      public static int[] freeman(point p1, point p2){
            double d=Math.sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-
p2.y));
            double angle;
            double cosa = (p2.x-p1.x)/d;
            double sina = (p2.y-p1.y)/d;
            if (sina >=0) angle=Math.acos(cosa);
            else angle=2*Math.PI-Math.acos(cosa);

            if (angle <= Math.PI*1/8){ int[] f={0,1,0}; return f;}
               //{valeur de freeman, direction en x, direction en y}
            if (angle <= Math.PI*3/8){ int[] f={1,1,1}; return f;}
            if (angle <= Math.PI*5/8){ int[] f={2,0,1}; return f;}
            if (angle <= Math.PI*7/8){ int[] f={3,-1,1}; return f;}
            if (angle <= Math.PI*9/8){ int[] f={4,-1,0}; return f;}
            if (angle <= Math.PI*11/8){   int[] f={5,-1,-1}; return f;}
            if (angle <= Math.PI*13/8){   int[] f={6,0,-1}; return f;}
            if (angle <= Math.PI*15/8){   int[] f={7,1,-1}; return f;}
            int[] f={0,1,0}; return f;
      }

      /*********************************************************/

      //normalisation de la taille de l'enveloppe
      public static void norme(point[] p){
            int xmin=p[0].x;  // hauteur ramenée entre 0 et 100;
                                    // longueur à partir de 0 dilatée
proportionnellement
            int xmax=p[0].x;
            int ymin=p[0].y;
            int ymax=p[0].y;
            for (int i=0;i<p.length;i++){
```

```
                  if (p[i].x<xmin) xmin=p[i].x;
                  if (p[i].x>xmax) xmax=p[i].x;
                  if (p[i].y<ymin) ymin=p[i].y;
                  if (p[i].y>ymax) ymax=p[i].y;
            }
            for (int i=0;i<p.length;i++){
                  p[i].x=(int)(100/(float)(xmax-xmin)*(p[i].x-xmin));
                  p[i].y=(int)(100/(float)(xmax-xmin)*(p[i].y-ymin));
            }
      }

      /*********************************************************/

      //redresse l'enveloppe pour la rendre horizontale
      public static void redresse(point[] p){
            int dmax2=0;
            double cosa=0;
            double sina=0;


            //angle de la plus grande distance entre les pts de l'enveloppe
            for (int i=0;i<p.length;i++){
                  for(int j=i+1;j<p.length;j++){
                        int d2=(p[i].x-p[j].x)*(p[i].x-p[j].x)+(p[i].y-
p[j].y)*(p[i].y-p[j].y);
                        if (dmax2<d2) {
                              dmax2=d2;
                              sina=(p[i].y-p[j].y)/Math.sqrt(d2);
                              cosa=(p[i].x-p[j].x)/Math.sqrt(d2);
                        }

                  }
            }

            //rotation des points
            for (int i=0;i<p.length;i++){
                  double x=p[i].x;
                  double y=p[i].y;
                  if (cosa>=0){
                        p[i].x=(int)(cosa*x+sina*y);
                        p[i].y=(int)(-sina*x+cosa*y);
                  }
                  else{
                        p[i].x=(int)(-cosa*x-sina*y);
                        p[i].y=(int)(sina*x-cosa*y);
                  }
            }
      }

}

/*********************************************************/

class comparaison{

      //comparaison de 2 chaines de freeman, coutdroit= cout d'insertion-
suppression
      public static int comparaison(int[] seq1, int lseq1, int[] seq2 ,int
lseq2, int coutdroit) throws IOException{

            int[][] t= new int[lseq1][lseq2];
            t[0][0]=0;

            for (int i=1;i<lseq1;i++){
```

```
                  t[i][0]=t[i-1][0]+coutdroit;
            }
            for (int j=1;j<lseq2;j++){
                  t[0][j]=t[0][j-1]+coutdroit;
            }

            for (int i=1;i<lseq1;i++){
                  for (int j=1;j<lseq2;j++){
                        int coutdiag=Math.min(Math.abs(seq1[i]-seq2[j]),8-
Math.abs(seq1[i]-seq2[j]));
                        t[i][j]=min(t[i-1][j]+coutdroit, t[i]
[j-1]+coutdroit, t[i-1][j-1]+coutdiag);
                  }
            }

            //possibilité d'éditer le tableau de programmation
            /*BufferedWriter pd =new BufferedWriter(new
FileWriter("./txt/progdyn.txt"));
            for (int i=0;i<lseq1;i++){
                  for (int j=0;j<lseq2;j++){
                        pd.write(t[i][j]+"\t");
                        System.out.print(t[i][j]+" ");
                  }
                  pd.write("\r");
                  System.out.println("");
            }
            pd.flush();*/

            return t[lseq1-1][lseq2-1];
      }

      /*********************************************************/

      public static int min(int m1, int m2, int m3){
            int m = Math.min(m1,m2);
            m = Math.min(m3,m);
            return m;
      }

}

/*********************************************************/
```

Rdf.java

```
import java.awt.*;
import java.awt.image.*;
import javax.swing.*;
import java.net.URL;

public class Rdf extends JFrame {

      public point[] algorithm(BufferedImage input){
            int width = input.getWidth(this);
            int height = input.getHeight(this);

            //Convert the image in a pixel table
            int[] pixels = new int[width*height];
            PixelGrabber pg = new
PixelGrabber(input,0,0,width,height,pixels,0,width);
            try {
                  pg.grabPixels();
            } catch (InterruptedException e) {
```

```
            System.out.println("Interrupted waiting for pixels...");
        }

        //Treat the image
        //REM : THE IMAGE MUST BE A TWO COLOR IMAGE

            //Determine the int value of the two points

        int mini = Integer.MAX_VALUE;
        int maxi = -Integer.MAX_VALUE;
        for (int j=0; j<height; j++) {
            for (int i=0; i<(width); i++) {
                int cur = (j * width)+i;
                int pixel = pixels[cur];

                mini=Math.min(mini,pixel);
                maxi=Math.max(maxi,pixel);
            }
        }

            //Find the number of points
        int pointnumber=0 ;
        for (int j=0; j<height; j++) {
            for (int i=0; i<(width); i++) {
                int cur = (j * width)+i;
                int pixel = pixels[cur];

                if (pixel==mini) {
                    pointnumber++;
                }
            }
        }
            //Make the point table

        point[] points=new point[pointnumber];
        int k=0;
        for (int j=0; j<height; j++) {
            for (int i=0; i<(width); i++) {
                int cur = (j * width)+i;
                int pixel = pixels[cur];

                if (pixel==mini) {
                    points[k]=new point(i,j);
                    k++;
                }
            }
        }
        //Là on peut redresser l'ensemble de points obtenus
        //codage.redresse(points);

        //Compute the ConvexHull
        int taille=Rdf.convexHullAlg(points,pointnumber);
        //System.out.println("Résultat convex hull:");

        point[] hullB=new point[taille];
        for (int i=0;i<taille;i++) {
            hullB[i]=points[i].copy();
            //System.out.println(points[i].x+"\t"+points[i].y);
            }

        return hullB;
    }

    public static float theta(point p1, point p2) {
```

```
        int dx=p2.x-p1.x;
        int dy=p2.y-p1.y;
        int ax=Math.abs(dx);
        int ay=Math.abs(dy);
        float t;

        if ((dx==0)&&(dy==0)) { return 360;} //CED

        else {
              t=((float) dy)/(ax+ay);
        }
        if (dx<0) {
              t=2-t;
        }
        else if (dy<0) {
              t=4+t;
        }
        return (((t*90)+90) %360);
}

public static int convexHullAlg(point[] p, int number) {

        //Find point with lower x
        int miny=0;
        for (int i=1; i<number; i++) {
              if (p[i].x<p[miny].x) {
                    miny=i;
              }
        }

        //Put P[miny] in first position
        point t = p[0].copy();
        p[0]  = p[miny];
        p[miny]     = t;

        //System.out.println("0"+" x="+p[0].x+" y="+p[0].y);


        //Find other points of convex hull
        int M=0; //Number of points in the convex hull minus 1
        float minimumAngle=0;
        float lowAngle = 0;
        float highAngle = 360;

        //While mini is better than miny
        int mini;
        do
         {
              //Find lowest Angle successor
              //increment M
              M++;

              lowAngle = minimumAngle;
              highAngle = 360;
              mini = 0;//miny;

              for (int i=0; i<number; i++) {
                    if (
                          (theta(p[M-1],p[i])>=lowAngle)
                          &&
                          (theta(p[M-1],p[i])<highAngle)
                          ) {
```

```
                        mini = i;
                        highAngle = theta(p[M-1],p[i]);
                  }
            }
            minimumAngle=highAngle;

            //Put p[mini] in position M+1
            if (mini!=0) {
                  t = p[mini];
                  p[mini]     = p[M];
                  p[M]  = t;
            }

      }
      while ( mini!=0);
      //return the size of the convex hull
      return M;
}


public BufferedImage loadImage(String name) {
      URL url = Rdf.class.getResource(name);
      Image image = this.getToolkit().getImage(url);
      try {
                  MediaTracker tracker = new MediaTracker(this);
                  tracker.addImage(image, 0);
                  tracker.waitForID(0);
            } catch (Exception e) {}
            int width = image.getWidth(this);
            int height = image.getHeight(this);

            BufferedImage input = new BufferedImage(
                        width, height, BufferedImage.TYPE_INT_RGB);

            input.createGraphics().drawImage(image,0,0,null);
            return input;
}

public BufferedImage paint(BufferedImage b, point[] p){

      Graphics2D g2 = b.createGraphics();
      g2.setColor(Color.green);
      for   (int i=0; i<(p.length-1); i++) {
            //System.out.println(p[i].x+"\t"+p[i].y);
            g2.drawLine(p[i].x,p[i].y,p[i+1].x,p[i+1].y);
      }
      g2.drawLine(p[p.length-1].x,p[p.length-1].y,p[0].x,p[0].y);
      return b;
      }

public static void main(String[] args) {
      System.out.println("Execution Rdf");
      //Load
      Rdf eg = new Rdf();
      BufferedImage input= eg.loadImage("./data/chambre2.gif");

      //Compute
      point[] convexHull=eg.algorithm(input);
      //Print
      input = eg.paint(input, convexHull);

      System.out.println("Milieu Execution Rdf");

      //Display
```

```
            ConvexHullCanvas chc=new ConvexHullCanvas();
            eg.getContentPane().add(new JLabel(new ImageIcon(input)));
            eg.pack();
            eg.setVisible(true);
      }
}
/*********************************************************/
```